UNIT 5

BACKTRACKING

Patricia Cuesta Ruiz

Isabel Blanco Martínez

EXERCISE 4

A board M of size RxC is available (R is the number of rows and C the number of columns) and a chess horse is put in a starting square (posx, posy). The objective is to find, if possible, the way in which the horse must move around the board so that each box is used only once in the course (the 8x8 board always has a solution regardless of where the horse starts). The horse can finish in any position on the board.

A horse has eight possible movements (assuming, of course, that it does not leave the board). A movement between the squares M_{ij} and M_{pq} is valid only if:

- (|p-i|=1) && (|q-j|=2), or if
- (|p-i|=2) && (|q-j|=1),

that is, a coordinate changes two units and the other a single unit.

Code:

```python
import math

rows = 6
columns = 6
ini = 0
end = 0
total = rows * columns

def create_board():
    board = []
    for _ in range (rows):
        aux = [0] * columns
        board.append(aux)
    return board

def is_valid(row1, column1, row2, column2):
    abs1 = abs(row2 - row1)
    abs2 = abs(column2 - column1)
    return (((abs1 == 1) and (abs2 == 2)) or ((abs1 == 2) and (abs2 == 1)))

def print_board(board):
    for i in range (len(board)):
        print(board[i], end = "\n")

def movement(board, row, column, cont):
    if (cont >= rows * columns):
        print_board(board)
        return True
    for i in range (rows):
        for j in range (columns):
            if (is_valid(row, column, i, j) and (board[i][j] == 0)):
                board[i][j] = "x"
                exito = movement(board, i, j, cont + 1)
                board[i][j] = 0
                if (exito):
                    print("Movement to", i, ",", j , "\n")
                    return True

board = create_board()
board[ini][end] = "X"
movement(board, ini, end, 1)
```

Output:

```
['X', 'x', 'x', 'x', 'x', 'x']
['x', 'x', 'x', 'x', 'x', 'x']
['x', 'x', 'x', 'x', 'x', 'x']
['x', 'x', 'x', 'x', 'x', 'x']
['x', 'x', 'x', 'x', 'x', 'x']
['x', 'x', 'x', 'x', 'x', 'x']
Movement to 5 , 4

Movement to 3 , 5

Movement to 1 , 4

Movement to 3 , 3

Movement to 2 , 5

Movement to 4 , 4

Movement to 5 , 2

Movement to 4 , 0

Movement to 2 , 1

Movement to 4 , 2

Movement to 5 , 0

Movement to 3 , 1

Movement to 4 , 3

Movement to 5 , 5

Movement to 3 , 4

Movement to 1 , 5

Movement to 0 , 3

Movement to 1 , 1

Movement to 3 , 0

Movement to 5 , 1

Movement to 3 , 2

Movement to 2 , 0

Movement to 4 , 1
```

```
Movement to 5 , 3

Movement to 4 , 5

Movement to 2 , 4

Movement to 0 , 5

Movement to 1 , 3

Movement to 0 , 1

Movement to 2 , 2

Movement to 1 , 0

Movement to 0 , 2

Movement to 2 , 3

Movement to 0 , 4

Movement to 1 , 2
```

Explication:

This exercise is very similar to exercise 5. We have a board with dimensions row*columns where rows = columns = N. On the board we have a horse that has to step on every box, but it can't step on the same box twice.

We have implemented the function create_board to get the board initialized with the given rows and columns. All the boxes of the board start as 0.

We have another function is_valid that checks that the movement that wants to be realized is valid, following the rules given by the exercise (the movement is always an L).

The function print_board is in charge of printing the result of the board after being all the boxes visited.

The function movement is in charge of the backtracking. Using a counter, we will be able to know the total boxes that have been visited and when the counter is the maximum possible, the resulting board will be printed. To visit a box, we have to check if it is a valid one, and if it has not already been visited. If it is possible to visit it, its content happens to be 'x'. If not, its content is '0' again and we try with another box.

The position where the horse starts is shown as an 'X', to distinguish among the rest of 'x'.

After printing the board, all the movements that the horse has made are indicated.

EXERCISE 6:

You have the substitution table that appears below:

	a	b	c	d
a	b	b	a	d
b	c	a	d	a
c	b	a	c	c
d	d	c	d	b

which is used in the following way: in any string, two consecutive characters can be replaced by the value that appears in the table, using the first character as row and the second character as column. For example, you can change the sequence ca to a b, since M[c, a]=b.

Implement a Backtracking algorithm that, starting from a string of text and using the information stored in a substitution table M, is able to find a way to make the substitutions that allow reducing the text string to a final character, if possible.

Example: With the string text = acabada and the final character = d, a possible form of substitution is the following (the sequences that are replaced are marked for clarity): aca*ba*da → a*ca*cda → abc*da* → *ab*cd → b*cd* → *bc* → d.

Code:

```
def substitution(table, c1, c2):
    return table[index[c1]][index[c2]]

def algorithm(table, word, final):
    newChar = ""
    aux = ""
    listAux = []
    if (len(word) == 1):
        if (word == final):
            print("The objective has been reached: " + word)
            return True
        else:
            return False
    else:
        for i in range (len(word) - 1):
            newChar = substitution(table, word[i], word[i+1])
            aux = word.replace((word[i] + word[i+1]), newChar)
            if aux not in listAux:
                listAux.append(aux)
            aux = ""
        for j in range (len(listAux)):
            if (algorithm(table, listAux[j], final)):
                print("Path until the objective " + str(listAux[j]))
                return True
            else:
                algorithm(table, listAux[j], final)

table = [['b', 'b', 'a', 'd'],
         ['c', 'a', 'd', 'a'],
         ['b', 'a', 'c', 'c'],
         ['d', 'c', 'd', 'b']]
index = {'a': 0, 'b': 1, 'c': 2, 'd': 3}
word = 'acabada'
final = 'd'
algorithm(table, word, final)
```

Output:

```
The objective has been reached: d
Path until the objective d
Path until the objective da
Path until the objective ada
Path until the objective aada
Path until the objective bbada
Path until the objective aabada
```

Explanation:

In order to solve the exercise, we will use a matrix two-dimensional, where the first ever position is referenced as [0][0].

We are using a python dictionary in order to interrelate 0, 1, 2, 3 to the keys a, b, c, d.

In the function substitution we are passing the table, character1 and character2. It gives us the resultant character after substituting the two characters using the dictionary to find the necessary position of the table.

The function algorithm is in charge of exploring all the possible paths until getting the initial objective defined, using auxiliar strings where we store the new string with the new substituted character. We also use a list to store the path chosen and print it in the output. When the length of the string is 1 and it is the same as the objective provided, it finishes, and the final string is shown.

If the final objective is not achieved, it goes back and tries to find new paths.